Einführung in die Programmierung des NXT mit C

1. Einführung zu Lego Mindstorms und zur Programmierung in C (NXC)

- Einführung
- Entwicklungsumgebung
- „Hallo Welt“ auf dem NXT
- Bau des Roboters

2. Grundlegende Elemente der Programmiersprache NXC
- Tasks
- Variablen
- bedingte Anweisungen
- Schleifen

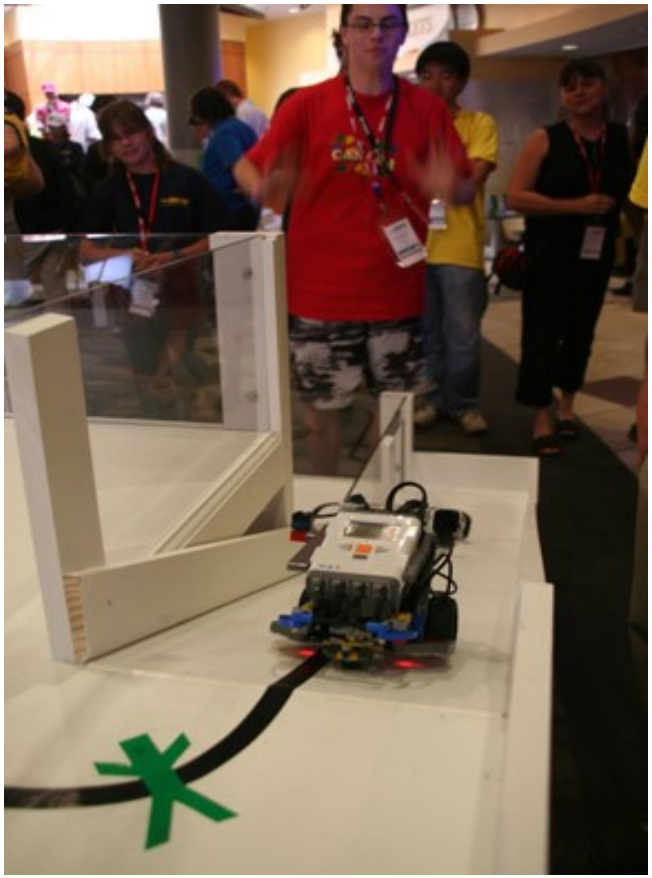
3. Motoren
- API
- Übungsaufgaben

4. Sensoren
- API
- Übungsaufgaben

5. Multitasking

Projekt:

- Programmierung eines Linienfolgers



Literatur / Links

NXT Power Programming, John Hansen
ISBN:978-0-9738649-2-2

Entwicklungsumgebung “Bricx Command Center (BricxCC)”
http://bricxcc.sourceforge.net/

Programmiersprache NXC
http://bricxcc.sourceforge.net/nbc/

Tutorial und Guide zu NXC (als pdf zum Ausdrucken)
http://bricxcc.sourceforge.net/nbc/nxcdoc/index.html

Roboter

Roboter

- *P. Hoppen: Autonomer mobiler Roboter*
  „Maschine, die sich in einer natürlichen Umgebung aus eigener Kraft und ohne Hilfestellung von außen bewegen und dabei ein ihr gestelltes Ziel erreichen kann. […] Dabei erkennt sie die Umwelt, sofern dies notwendig ist, über eigene Sensoren.“

Einteilung von Robotern nach Einsatzgebiet:

- Industrieroboter (stationär, nicht autonom, z.B. Schweißroboter)
- Serviceroboter (Putzroboter, Museumsführer)
- Geländeroboter (Einsatz in schwer zugänglichen oder gefährlichen Bereichen)

Motivation für mobile Roboter

- Traum vom menschenähnlichen Helfer
- Lösung gefährlicher oder unangenehmer Aufgaben
- Spiele

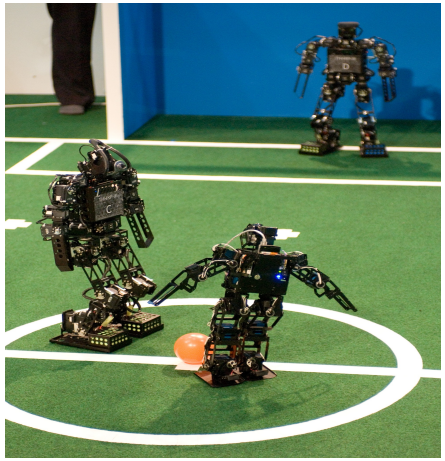
Mobile Roboter stellen durch die sich ändernde Umwelt eine Herausforderung an die Informationsverarbeitung dar.

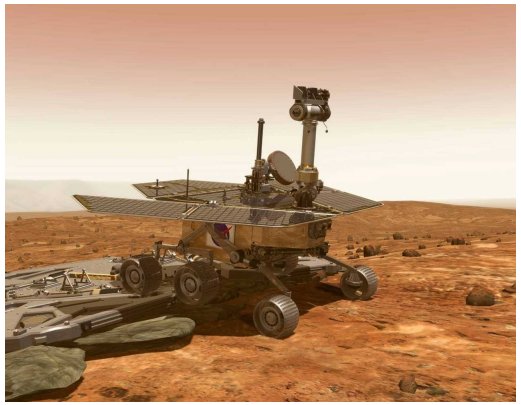
Anwendungen: ??

Anwendungen:

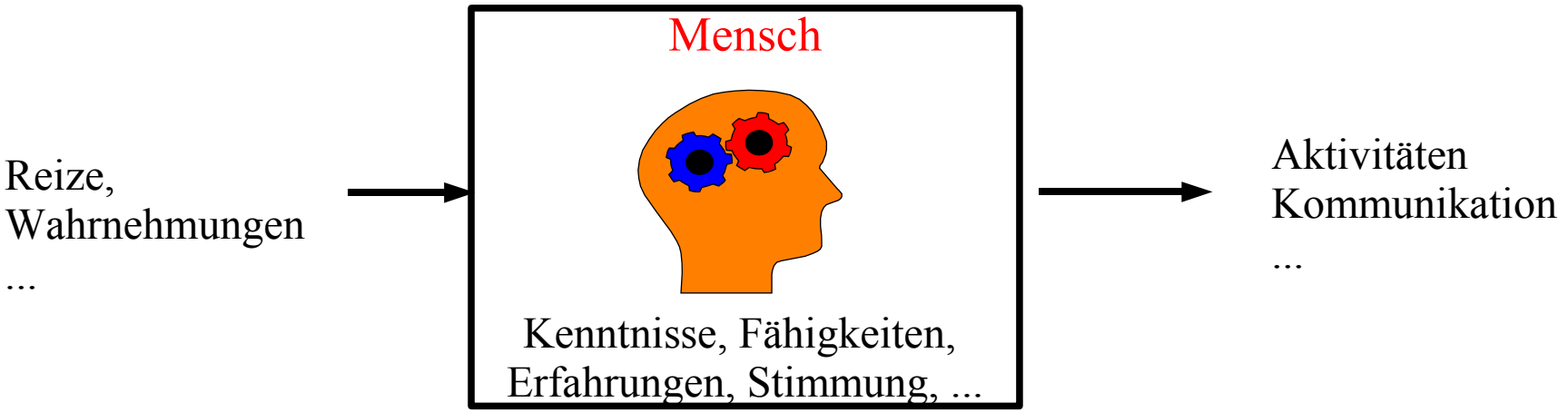
- Haushalts- und Servicerobotik
- Fahrerlose Transportsysteme in der Produktion

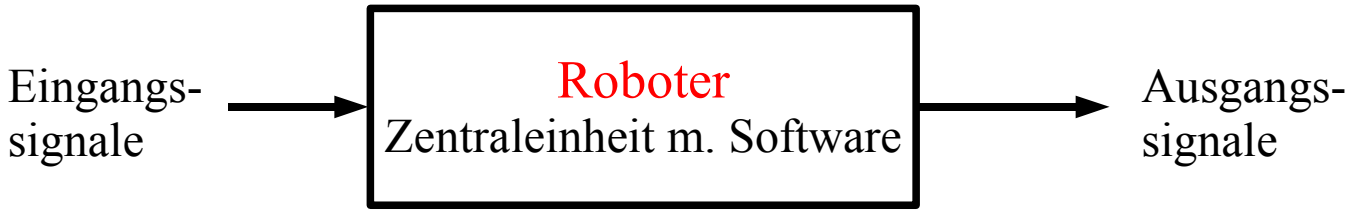
- Einsatz von Robotern nach Katastrophen
- Roboterfußball im RoboCup





Grundlegende Funktionsweise von Robotern





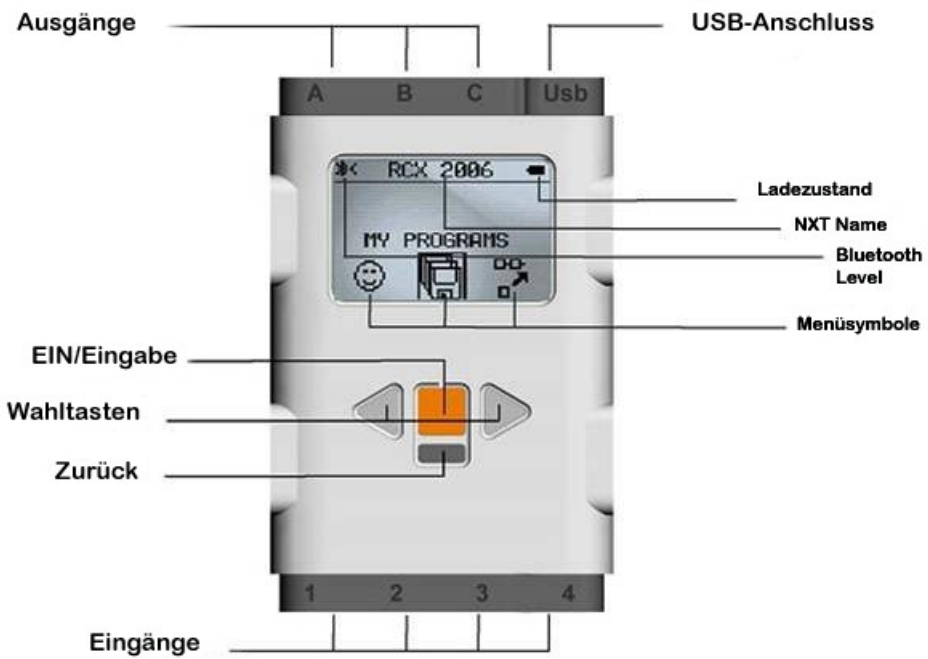
Video vom Robocup zeigen!!! http://www.robocup-german-open.de/de/videos

Roboter mit LEGO MINDSTORMS

NXT-Baustein (Zentraleinheit)

- Microprozessor: Atmel ARM, 32 Bit, 48 Mhz
- 256 kByte Flash
- 64 kByte RAM
- Koprozessor: Atmel 8-Bit AVR 8 MHz 4 KB Flash-Speicher, 512 Byte RAM,
- 4 Eingänge für Sensoren, davon 1 Highspeed mit 926 Kbit/s
- 3 Ausgänge für Motoren
- Bluetooth
- USB-2.0-Anschluss, 12 Mbit/s
- LCD-Anzeige; 100 × 64 Pixel
- Soundausgabe
- 6 x AA-Batterie oder Akku
- Lithium-Akkupack von LEGO erhältlich

Tasten und Anschlüsse des NXT



Funktion „View“ erklären!!

Programmieren mit NXC

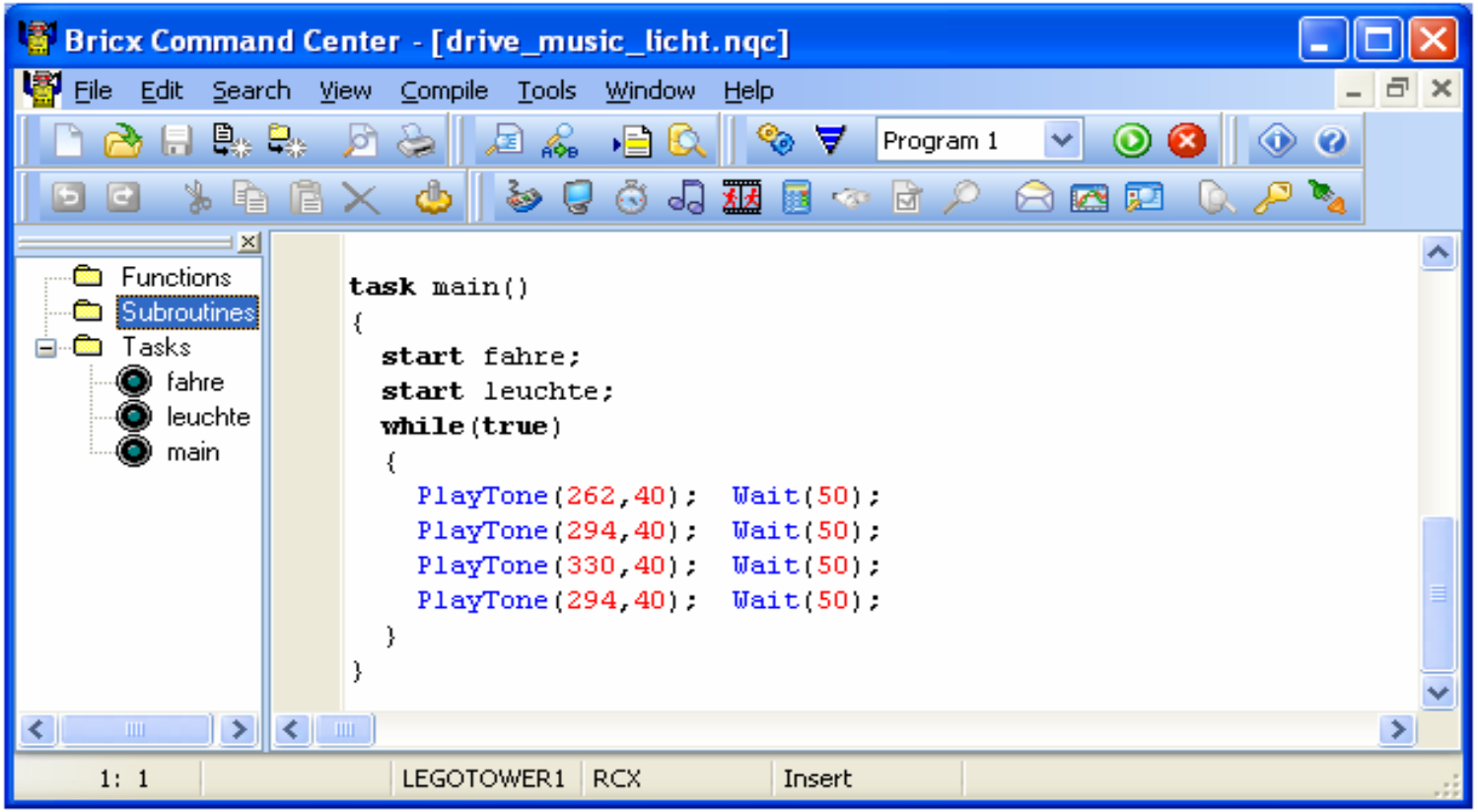
- C-ähnliche Programmiersprache :
  „Not eXactly C“
- Weiterentwicklung von NQC für den RCX
- Entwickler: John Hansen (Softwareingenieur aus den USA)
- Compiler und Entwicklungsumgebung sind freie Software und Open Source

Ablauf bei der Entwicklung von NXC-Programmen:

1. Konzept für die Problemlösung erarbeiten (Schritte des Algorithmus')
2. Programm in NXC schreiben
3. Programm übersetzen, laden und testen

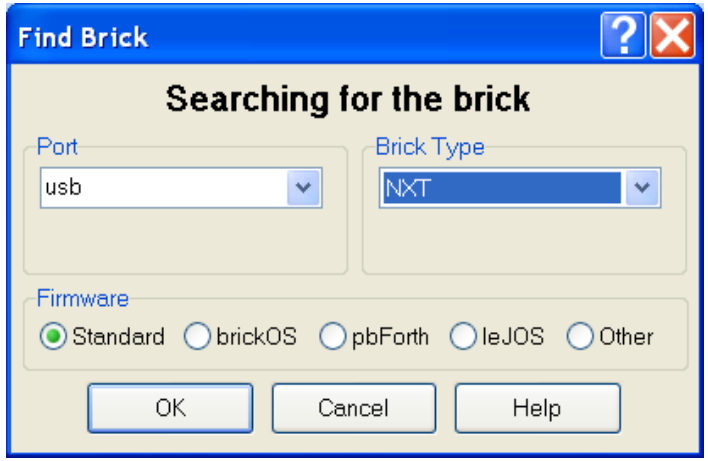
Programmentwicklung mit BricxCC (Bricx Command Center)

Entwicklungsumgebung für NQC-Programme
- erleichtert das Schreiben von Programmen und die Fehlersuche
- überträgt die übersetzten Programme an den Lego-Roboter

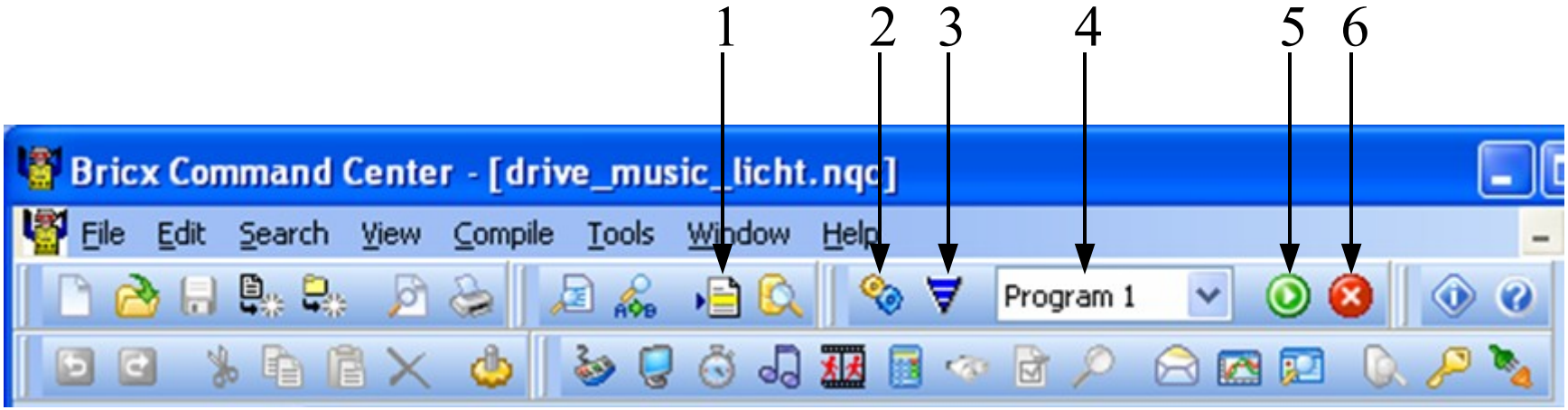


BricxCC – Starten und Verbindung herstellen

- Bei jedem Öffnen des BricxCC erscheint ein Dialogfenster:
  - der Port muss auf „usb“ eingestellt werden
  - danach auf „OK“ klicken
- Der Computer versucht mit dem NXT zu kommunizieren.
  - Dazu muss der NXT (Lego-Roboter) eingeschaltet sein und das USB-Kabel verbunden sein
  - Wenn kein NXT vorhanden ist, einfach auf „Cancel“ klicken, um das Programm zu öffnen. Allerdings sind dann einige Funktionen deaktiviert.



BricxCC – ausgewählte Funktionen



1. Gehe zu Zeile Nr.
2. **Programm nur kompilieren (für den NXT „übersetzten“)**
3. **Programm kompilieren und auf den NXT übertragen**
4. Programmplatz wählen (nur für RCX)
5. Programm starten (F7)
6. Programm stoppen (F8)

NXC – Not eXactly C

- Ein NXC-Hauptprogramm (Task) heißt immer main und enthält
  - Anweisungen (Befehle in C-ähnlicher Syntax) und
  - Kommentare (zur Erläuterung der Programme)

```
task main()
{
    //  Einzeilige Kommentare beginnen mit 2 Schrägstrichen.

    /* (mehrzeiliger Kommentar)
       Zwischen den geschweiften Klammern
       stehen die Anweisungen für den RCX.
    */
}
```

NXC – Not eXactly C

- NXC-Anweisungen (Befehle):
  - enden immer mit einem Semikolon
  - können einen oder mehrere Parameter enthalten
    - Eingänge
    - Ausgänge
    - Zeiten
    - ...
- Beispiel:

```
task main()
{     SetPower(OUT_A, 7); /* manche Befehle enthalten Parameter,
                             die durch Komma getrennt
                                 in runden Klammern stehen.*/
      OnFwd(OUT_A);          // Befehle enden immer mit Semikolon
}
```

„Hallo Welt“ auf dem NXT

```
//Hallo Welt

task main()
{
   TextOut(0, LCD_LINE1, "Hallo Welt");

  Wait(5000);
}
```

Darstellung von Compilerfehlern und Hilfefunktionen erklären!

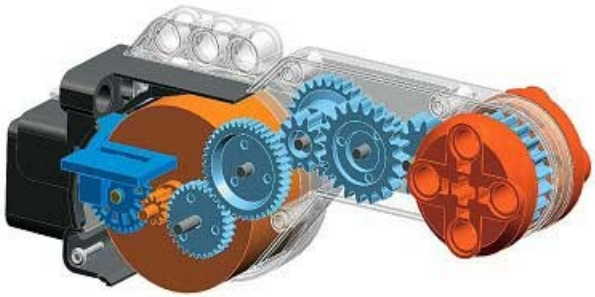
„Hallo Welt“ (erweiterte Version)

```
/*Hallo Welt (Version 2 mit Variable und
                Variablenausgabe auf dem LCD) */

task main()
{
  int version=2;

  TextOut(0, LCD_LINE1, "Hallo Welt");

  TextOut(0,LCD_LINE2, "Version: ");
  NumOut(50,LCD_LINE2,version);

  Wait(5000);
}
```

Bau eines einfachen Roboters

- Bauanleitung „Mindstorms Education“ Seite 8 ... 22

Motoren



- Getriebemotoren mit integriertem Inkrementaldrehgeber
- Werden von Koprozessor Atmel 8-Bit AVR 8 Mhz kontrolliert
- Bieten vielfältige Möglichkeiten besonders für odometrische Navigation

Motoren

- 3 Ausgänge für Motoren: **`OUT_A, OUT_B, OUT_C`**
  - Zustände: ein / aus- gebremst / aus- „floating“ (Motoren drehen sich frei)
  - Leistung und Richtung können eingestellt werden

- Programmierung:
  - Geschwindigkeit festlegen: 100 Stufen (0 .. 99)
  - Motor in bestimmter Richtung starten: vorwärts / rückwärts
  - Motor stoppen

Ansteuerung der Motoren

```
OnFwd(OUT_A, 75);   // Ausgang A  mit 75% vorwärts

OnRev(OUT_C, 50);   // Ausgang C mit 50% rückwärts

Off(OUT_AC);        // Ausgänge A und C ausschalten

OnFwdSync(OUT_AC, 75,0);  // Ausgang A und C  mit 75%
                          vorwärts geradeaus

OnFwdSync(OUT_AC, 50,100);  // Ausgang A und C  mit 50% auf
                               d. Stelle nach rechts drehen

OnFwdSync(OUT_AC, 50,-100);  // Ausgang A und C  mit 50% auf
                                d. Stelle nach links drehen
```

Übungsaufgabe 1 – Etwas Einfaches zum Anfang

Programmiere einen Roboter,
der geradeaus fährt und nach 2 Sekunden anhält.

Hinweis: Mit „Wait(2000)“ wird im Programm eine Pause eingelegt, eingeschaltete Motoren bleiben eingeschaltet.

Übungsaufgabe 2.1 – Drehen auf der Stelle

Programmiere einen Roboter, der sich auf der Stelle zunächst 1 Sekunde lang nach links dreht, dann 2 Sekunden nach rechts und wieder eine Sekunde nach links, so dass er am Ende wieder geradeaus schaut.

Hinweis: Das Drehen auf der Stelle wird durch gegenläufiges Rotieren der Räder realisiert.

Übungsaufgabe 2.2 – Um die Kurve

In einem zweiten Programm soll der Roboter eine Rechtskurve mit einem größeren Radius fahren.

Übungsaufgabe 3 – Quadrat fahren

Programmiere einen Roboter, der ein vollständiges Quadrat abfährt und dann anhält. Er soll immer im Wechsel erst vorwärts fahren und sich nach 2 Sekunden um ca. 90 Grad nach rechts drehen.

Wiederholung von Anweisungen mit Repeat-Schleifen

Wiederholte Ausführung einer Aktivität

Syntax:

```
repeat (anzahl)
anweisung1;
```

Wirkung:

- wiederhole **anweisung1** n-mal

Beispiel:

```
repeat (4) {// 4 x
   PlayTone(440, 25);      // Ton wiederholt ausgeben und
   Wait(25);               // eine kurze Pause machen
}
```

Übungsaufgabe 3.1 – Quadrat fahren mit Repeat-Schleife

Wie Aufgabe 3.1 , aber mit Schleife

Variablen- Gedächtnis des Roboters

Zum Merken von logischen und numerischen Werten eines bestimmten Datentyps auf einem benannten Speicherplatz.

Datentyp: Wertemenge, z. B. ist `int` die Menge der ganzen Zahlen und `bool` = {true, false} die Menge der logischen Werte.

Zur Verwendung einer Variablen muss diese **deklariert** werden:

Datentyp Name; (Der Name ist in gewissen syntaktischen Grenzen frei wählbar)

Beispiele: `int zaehler; bool fertig;`

Zuweisung eines passenden Werts an eine Variable

Variablenname = Berechnungsformel;

Die Berechnungsformel wird ausgewertet und der berechnete Wert der Variablen mit dem angegebenen Namen zugewiesen.

Beispiele:

- `zaehler = 0; // zaehler wird der Wert 0 zugewiesen`
- `zaehler = zaehler + 1; // zaehler wird um 1 erhöht`
- `fertig = !(zaehler < 4) // ! ist die logische Negation`

Übungsaufgabe 4 – größer werdendes Quadrat fahren

Programmiere einen Roboter, der 2 vollständige Quadrate fährt, wobei jede gerade Strecke, die er vorwärts fährt, etwas länger sein soll als die vorherige.

Hinweis: Benutze dafür eine Variable, die bei jedem Schleifendurchlauf erhöht wird.

Sensoren für den NXT

- NXT besitzt 4 Sensorports für I²C-Sensoren
  (I²C: serieller Datenbus 9600 Baud)
- Port 4 auch als Highspeed (926 kBaud) nutzbar (z.B. für Digitalkamera)

Sensoren von LEGO:
- Berührungssensor
- Lichtsensor
- Soundsensor
- Ultraschallsensor

Sensoren von Drittherstellern (Hitechnic, Mindsensors)
- Kompasssensor
- Farbsensor
- Beschleinigungssensor
- Drucksensor, IR-Sucher usw.

Sensor-API

- 4 Sensorports S1 ... S4 oder IN_1 .... IN_4
- können mit verschiedenen Sensor-Typen verbunden werden
- Sensorwerte stehen in den Variablen `SENSOR_1 .... SENSOR_2`
- können auf unterschiedliche Art (aufbereitet und) übergeben werden
- Betriebsarten (Modi) für Sensor-Eingänge:
  - Rohwert („Rohmodus“) - tatsächlich gemessener Wert als ganze Zahl
  - boolsche (logische) Werte („logischer Modus“ - Werte „0“ oder „1“
  - verarbeitete Werte – z.B. Prozentmodus, Celsius usw.

..

Sensoren benutzen

-Sensoreingänge müssen vor Benutzung konfiguriert werden:

```
SetSensorType (S1, SENSOR_TYPE_LIGHT_ACTIVE);
SetSensorMode (S1, SENSOR_MODE_PERCENT);
ResetSensor(S1);  // notwendig, sonst werden Einstellungen nicht
                  // übernommen
```

Sensor abfragen:

```
val=SensorValue(S1);
```

oder

```
val=SENSOR_1;
```

Ultraschall-Sensor, Farbsensor, IR-Sucher und andere von Drittherstellern besitzen eine andere API!

Weitere Sensortypen- und Modi

SENSOR_TYPE_TOUCH
SENSOR_TYPE_LIGHT_ACTIVE
SENSOR_TYPE_LIGHT_INACTIVE
SENSOR_TYPE_SOUND_DB
SENSOR_TYPE_SOUND_DBA
SENSOR_TYPE_ROTATION

SENSOR_MODE_RAW	raw value from 0 to 1023
SENSOR_MODE_BOOL	boolean value (0 or 1)
SENSOR_MODE_EDGE	counts number of boolean transitions
SENSOR_MODE_PULSE	counts number of boolean periods
SENSOR_MODE_PERCENT	value from 0 to 100
SENSOR_MODE_FAHRENHEIT	degrees F
SENSOR_MODE_CELSIUS	degrees C
SENSOR_MODE_ROTATION	

„Bedingte Anweisung“

Syntax:

```
if (bedingung) anweisung1; else anweisung2;
```

Wirkung:

- **Wenn** die **Bedingung wahr** ist, führe **anweisung1** aus,
- **ansonsten**, führe **anweisung2** aus

Typische Bedingungen:

- Test auf Gleichheit `SENSOR == wert`
- Test auf Ungleichheit `SENSOR != wert`
- Test auf Überschreitung `SENSOR > wert` oder `SENSOR >= wert`
- Test auf Unterschreitung `SENSOR < wert` oder `SENSOR <= wert`

Variante der Benutzung:

- **else** und **anweisung2** (else-Teil) kann man weglassen

Beispiel:

```
if (SENSOR_1 == 1)
      PlayTone(100, 1000);   //   tiefen Ton
else                        // ansonsten
      PlayTone(440, 1000);   //   hohen Ton
```

Zyklische Aktivitäten mit while-Schleifen

Wiederholte Ausführung einer Aktivität unter bestimmten Bedingungen

Syntax:

```
while (bedingung) anweisung1;
```

Wirkung:

- **Solange** die **Bedingung wahr** ist, führe **anweisung1** aus

Beispiel:

```
while (SENSOR_1 == 0) {   // solange Sensor 1 den Wert 0 hat
  PlayTone(440, 25);      // Ton wiederholt ausgeben und
  Wait(25);               // eine kurze Pause machen
}
// die folgende Anweisung wird erst dann ausgeführt,
// wenn der SENSOR_1 einen Wert verschieden von 0 hat
Off(OUT_A);               // Motor aussschalten
```

Sonderfall: Endlosschleife

```
while ( 1 )           // Wert <> 0 entpricht "wahr"
  anweisung;          // läuft endlos, bis zum Programmabbruch
```

Übungsaufgabe 5 – Berührungssensor

Baue den Berührungssensor an den Roboter.

Der Roboter soll solange fahren, bis er mit dem Touchsensor ein Hindernis erkennt und dann stehen bleiben.

Erweitere das Programm so, daß er dem Hindernis ausweicht.

Ultraschallsensor

Der Ultraschallsensor sendet Ultraschallwellen aus und empfängt das vom Hindernis reflektierte Ultraschallsignal. Aus der Zeitdifferenz zwischen Senden und Empfang wird die Entfernung berechnet.

Der Ultraschallsensor ist am I2C-Bus angekoppelt und besitzt einen eigenen Chip zur Signalverarbeitung und eine eigene API.

Der Sensor liefert ab ca. 6cm und bis ca. 250 cm gültige Werte.

```
SetSensorLowspeed(S1);   // Sensor initialisieren

x=SensorUS(S1);          // Entfernungswert  auslesen
```

Übungsaufgabe 5.1
Ultraschallsensor – Berührungslose Hindernisvermeidung

Baue den Ultraschallsensor so an den Roboter, daß der Sensor nach vorn zeigt.

Zur Vorbereitung laß Dir in einer Endlosschleife den Entfernungswert auf dem LCD ausgeben.

Der Roboter soll geradeaus losfahren.
Wenn die gemessene Entfernung zu einem Hindernis kleiner als 20cm beträgt, soll der Roboter dem Hindernis ausweichen und in eine andere Richtung fahren.

Übungsaufgabe 6 – Lichtsensor

Baue den Lichtsensor so an den Roboter, daß der Sensor nach unten zeigt.

Zur Vorbereitung laß Dir in einer Endlosschleife den Lichtsensorwert auf dem LCD ausgeben.

Der Roboter soll solange geradeaus fahren, bis er einen dunklen Untergrund erkennt.

Übungsaufgabe 7 – Balken zählen

Der Roboter soll geradeaus fahren und nach dem vierten dunklen Balken auf dem Untergrund anhalten.

Parallele Aktivitäten: Multitasking

Tasks können im Hauptprogramm `main` gestartet oder gestoppt werden:

```
// parallel laufendes Unterprogramm
task sensor_ueberwachung()
{
    while (true)
    {
        ...
    }
}

// Hauptprogramm
task main()
{
    // das Unterprogramm wird gestartet
    start sensor_ueberwachung;
    ...
    // und später wieder angehalten
    stop sensor_ueberwachung;
}
```

Übungsaufgabe 8 – Fahren und Melodie spielen

Der Roboter soll solange rückwärts fahren und eine Melodie abspielen, bis er gegen ein Hindernis stößt. Trifft er auf ein Hindernis, soll er die „Musik“ stoppen, 1 s rückwärts fahren,sich um 90° drehen.

Danach soll er wieder bis zu einem Hindernis geradeaus fahren und die Melodie abspielen.

Lösungen für Übungsaufgaben

```
//Aufgabe1   Geradeaus und nach 2 Sek. anhalten

task main()
{
 OnFwd(OUT_A, 75);
 OnFwd(OUT_C, 75);
 Wait(2000);
 Off(OUT_AC);
}
```

```
//Aufgabe2.1  Drehen auf der Stelle

task main()
{
 OnFwd(OUT_A, 50);     // 1 Sek. nach links
 OnRev(OUT_C, 50);
 Wait(1000);

 OnRev(OUT_A, 50);   // 2 Sek. nach rechts
 OnFwd(OUT_C, 50);
 Wait(2000);

 OnFwd(OUT_A, 50);    // 1 Sek. nach links
 OnRev(OUT_C, 50);
 Wait(1000);

 Off(OUT_AC);
}
```

```
//Aufgabe2.2   /Rechtskurve fahren

task main()
{
  OnFwd(OUT_A, 40);
  OnFwd(OUT_C,60);
  Wait(3000);

  Off(OUT_AC);
}
```

//Aufgabe 3 Quadrat fahren mit Schleife

```
task main()
{

  repeat (4)
    {  OnFwd(OUT_A, 50); //Geradeaus
       OnFwd(OUT_C,50);
       Wait(2500);

       OnRev(OUT_A, 50); //Drehen
       Wait(500);
    }

  Off(OUT_AC);
}
```

//Aufgabe 3 Variante Quadrat fahren mit Schleife - synchronisierte Motoren

```
task main()
{

 const int anzahl=4;
 int i=1;

 while (i<= anzahl)
 {
  OnFwdSync(OUT_AC, 50,0); //Geradeaus
  Wait(2500);

  OnFwdSync(OUT_AC, 50,100); //Drehen mit 100%, d.h. auf der Stelle
  Wait(650);
  i++;
 }

 Off(OUT_AC);
}
```

//Aufgabe 4 Quadrat fahren mit größer werdenden Kantenlängen

```
task main()
{

    const int anzahl=8;
    const int faktor=500;
    int Anfangszeit=2500;

 repeat ( anzahl)
 {
   OnFwdSync(OUT_AC, 50,0); //Geradeaus
   Wait(Anfangszeit);
   Anfangszeit+=faktor;

   OnFwdSync(OUT_AC, 50,100); //Drehen mit 100%, d.h. auf der Stelle
   Wait(650);
}

 Off(OUT_AC);
}
```

```
//Aufgabe5   Geradeaus und nach Sensorauslösung anhalten bzw. ausweichen

task main()
{ int val;

 SetSensorType (S1, SENSOR_TYPE_TOUCH);
 SetSensorMode (S1, SENSOR_MODE_BOOL);
 ResetSensor(S1);

 OnFwd(OUT_AC, 75);

 while ( true)
 {
  if (SENSOR_1 ==1)
  {
   OnRev(OUT_AC,75); Wait(300);
   OnFwd(OUT_A,75); Wait(300);
   OnFwd(OUT_AC,75);
  }
 }
 }
```

```
//Aufgabe 5.1 Hindernisvermeidung mit
  Ultraschallsensor

#define NEAR 20 //cm

task main()
{
 SetSensorLowspeed(S1);
 while(true){
   OnFwd(OUT_AC,50);
   while(SensorUS(S1)>NEAR)
     Wait(10);
   Off(OUT_AC);
   OnRev(OUT_C,100);
   Wait(800);
 }
}
```

//Aufgabe 6 Geradeaus und beim schwarzen Balken anhalten

```
const int schwellwert=40;

task main()
{ int val;

 SetSensorType (S1, SENSOR_TYPE_LIGHT_ACTIVE);
 SetSensorMode (S1, SENSOR_MODE_PERCENT);
 ResetSensor(S1);

 OnFwd(OUT_A_C, 75);
 val=SensorValue(S1);
 while ( val > schwellwert )
 {
   val=SensorValue(S1);
   NumOut(0, LCD_LINE1,val);
 }

 Off(OUT_AC);
}
```

```
// Aufgabe 7  Geradeaus und schwarze Balken zählen (einfache Version)

int schwellwert=40;
int anzahl_balken=4;
task main()
{
 int val;
 int zaehler=0;
 SetSensorType (S1, SENSOR_TYPE_LIGHT_ACTIVE);
 SetSensorMode (S1, SENSOR_MODE_PERCENT);
 ResetSensor(S1);
 OnFwd(OUT_BC, 50);
 while (zaehler<anzahl_balken) {
  val=SensorValue(S1);
  if (val < schwellwert)
  {zaehler = zaehler + 1;
   PlayTone(1000, 100);
   Wait(300); // jeden Balken nur einmal berücksichtigen
  }
 }
 Off(OUT_BC);
}
```

```
// Aufgabe 8 Fahren mit Hindernisvermeidung und Multitasking
bool stopped;


task main()
{
 SetSensorType(S3, SENSOR_TYPE_TOUCH);
 SetSensorMode(S3, SENSOR_MODE_BOOL);
 ResetSensor(S3);
 stopped=false;

 start melody;

 while(true)
 {
  OnRevSync(OUT_BC,50,0);       //Rückwärts fahren
  int wert=SensorValue(S3);
```

```
   if ( wert==1)
      { stopped=true;
        OnFwd(OUT_BC,50);    //Vorwärts
        Wait(1000);
        OnFwdSync(OUT_BC,50,100); // Drehen
        Wait(300);
        stopped=false;
      }
   }
  }

task melody()
  {  while (true)
     { if (stopped)
         PlayTone(500,200);
       Wait(20);
     }
   }
```

//Aufgabe 7 Geradeaus und schwarze Balken zählen mit Multitasking

```
 const int schwellwert   =40;
 const int anzahl_balken =5;
 bool fertig;

task drive();
task ueberwache_sensor();

task main()
{
 fertig=false;
 start drive;
 start ueberwache_sensor;
}

 task drive()
 {
  OnFwd(OUT_AC, 50);
  while (fertig==false)
        Wait(10);
  Off(OUT_AC);
 }
```

```
task ueberwache_sensor()
{ int val;
  int zaehler=0;

 SetSensorType (S1, SENSOR_TYPE_LIGHT_ACTIVE);
 SetSensorMode (S1, SENSOR_MODE_PERCENT);
 ResetSensor(S1);  // notwendig, sonst werden Einstellungen nicht übernommen

 while (fertig==false)
  {
   val=SensorValue(S1);
   if (val < schwellwert )
      { zaehler++;
        PlayTone(1000, 100);
       }
   if(zaehler ==anzahl_balken)
      fertig=true;
  // NumOut(0, LCD_LINE1,val);
  Wait(200);
 }
}
}
```